arXiv:1506.05561v1 [cs.CL] 18 Jun 2015

Comparing and evaluating extended Lambek calculi

Richard Moot[*]
CNRS (LaBRI)
University of Bordeaux

June 19, 2015

Abstract

Lambeks Syntactic Calculus, commonly referred to as the Lambek calculus, was innovative in many ways, notably as a precursor of linear logic. But it also showed that we could treat our grammatical framework as a logic (as opposed to a logical theory). However, though it was successful in giving at least a basic treatment of many linguistic phenomena, it was also clear that a slightly more expressive logical calculus was needed for many other cases. Therefore, many extensions and variants of the Lambek calculus have been proposed, since the eighties and up until the present day. As a result, there is now a large class of calculi, each with its own empirical successes and theoretical results, but also each with its own logical primitives. This raises the question: how do we compare and evaluate these different logical formalisms?

To answer this question, I present two unifying frameworks for these extended Lambek calculi. Both are proof net calculi with graph contraction criteria.

The first calculus is a very general system: you specify the structure of your sequents and it gives you the connectives and contractions which correspond to it. The calculus can be extended with structural rules, which translate directly into graph rewrite rules.

The second calculus is first-order (multiplicative intuitionistic) linear logic, which turns out to have several other, independently proposed extensions of the Lambek calculus as fragments.

I will illustrate the use of each calculus in building bridges between analyses proposed in different frameworks, in highlighting differences and in helping to identify problems.

Keywords: type-logical grammar; Lambek calculus; proof nets; linear logic

1 Introduction

The Lambek calculus (Lambek, 1958) is a landmark formal system. It is a logically simple system, with a transparent interface to natural language semantics (van Benthem, 1987) and gives a basic account of some interesting phenomena on the syntax-semantics interface, such as quantifiers and extraction. However, researchers working on the Lambek calculus soon suspected that there were problems with the calculus, both of a formal and of a descriptive nature.

[*]This work has benefitted from the generous support of the French agency Agence Nationale de la Recherche as part of the research project Polymnie (ANR-12-CORD-0004)

Though this was only proved in (Pentus, 1995), it has long been suspected that Lambek grammars generate only context-free languages and there are compelling arguments that natural languages require at least a slightly larger class of languages (Shieber, 1985; Joshi, 1985).

On the descriptive side, the Lambek calculus handles quantifiers and extraction only from peripheral positions: we can approximate medial extraction and medial quantifiers taking wide scope only by multiplying and complicating the lexical entries. Many other phenomena are problematic for the Lambek calculus.

Besides these problems of undergeneration, it has been known since (Lambek, 1961) that the Lambek calculus suffers from overgeneration as well. Lambek lists several problem cases, but the global availability of the associativity rule means that we predict very bad coordinations such as "*The mother of and John thinks that Bill left"[1]. Lambek's (1961) non-associative Lambek calculus, NL, solves these problems of overgeneration but in doing so makes the system too restrictive in other ways. For example the simple treatment of peripheral extraction and quantification of the Lambek calculus is no longer available in NL.

One of the main research goals in type-logical grammars has been to keep the good properties of the Lambek calculus (notably its logical simplicity and its connection to natural language semantics) while solving its empirical problems accounting for linguistic data.

In pursuit of this goal, a rather large number of logics has been proposed, which include the Lambek-Grishin calculus (LG) (Bernardi and Moortgat, 2010), the Displacement calculus (Morrill et al., 2011), multimodal categorial grammars (Moortgat, 1996c), lambda grammars (Oehrle, 1994), NL_λ (Barker and Shan, 2014), Hybrid Type-Logical Grammars (Kubota and Levine, 2012), etc. All of these logics are a shift of perspective with respect to the original Lambek calculus: for example, the Displacement calculus changes the basic objects from strings to string tuples (thereby making it natural to add logical operations for infixes and circumfixes), whereas Hybrid Type-Logical Grammars add the lambda calculus term constructors of abstraction and application (and the associated reduction operations) to the Lambek calculus connectives.

There is a rather large number of logical calculi available all claiming to improve upon the Lambek calculus in one way or another. However, this proliferation of calculi makes it hard to compare and evaluate the benefits of different calculi with respect to one another.

Given the variety of logical primitives used in these different logics, we should not expect a single "silver bullet" which relates all these different calculi. However, I discuss two "meta-logics" which at least provide two general strategies for doing such comparisons and give some examples of how to apply them.

The rest of the paper in structured as follows. I will first briefly discuss the common core shared by type-logical grammars: multiplicative intuitionistic linear logic as the syntax-semantics interface, followed by a very brief discussion of some of the phenomena in syntax and the syntax-semantics interface which we want our type-logical grammars to treat.

Then, I will present two meta-logics both employing a variant of Danos's (1990) graph contractions. The first system is a general and flexible proof net system which can be adapted to different connectives and different structural rules. The second system is first-order linear logic and it has several other type-logical grammars as a special case.

[1]Paul Dekker was the first to note this example which has been mentioned in the literature since the early 90s. Under global associativity and standard lexical assignments, both "the mother of" and "John thinks that" are expressions of type $(s \,/\, (np \setminus s)) \,/\, np$.

In what follows, I will assume the reader is familiar with the Lambek calculus and has some basic knowledge of formal semantics and of proof theory.

2 Multiplicative intuitionistic linear logic and the syntax-semantics interface

One standard architectural choice is shared between the different type-logical grammars discussed in this paper: the syntax-semantics interface takes the form of a homomorphism from proofs in the given source logic to proofs in multiplicative intuitionistic linear logic[2] (also called the Lambek-van Benthem calculus LP, in the categorial grammar literature).

In what follows, we will call the multiplicative intuitionistic linear logic proof the *deep structure* of a sentence and the proof in the source logic the *surface structure*. From the deep structure we obtain the meaning of a sentence by substituting the semantic terms from the lexicon and normalising the result term.

Seen from this perspective, different type-logical grammars generally agree on the deep structure for an analysis, but arrive at this deep structure from different surface structures. As Moortgat (1997, Section 3) remarks, the tension between the LP proofs of deep structure and the more restricted proofs of surface structure has played an important role in the development of type-logical grammars. We have to find a delicate balance between avoiding overgeneration and generating all the desired readings of our sentences.

3 A catalogue of problem cases

What are the linguistic data which motivated these new calculi? The following list gives some of the types of phenomena people have look at. Since a finite list of cases can always be treated by some additional lexical type assignments, we will be interested only in *robust* solutions, that is solutions which generalise beyond the listed examples to more complex cases, and ideally to different *phenomena*. In other words, we want to avoid ad hoc solutions (though, as an illustration, I will sometimes give an additional assignment which solves a particular case).

As usual, an asterisk "*" before a sentence denotes ungrammaticality.

(1) John loved but Peter hated "Syntactic Structures".

(2) *Peter bough the book which John read "Syntactic Structures" and Mindy liked.

(3) Peter bough the book which John read yesterday.

(4) John believes someone left.

(5) (dat) Jan Henk Marie de nijlpaarden zag helpen voeren.

(6) John studies logic and Charles, phonetics.

(7) John left before Mary did.

(8) Mary gave more girls a book than boys a record.

[2]The alternative is to use a simple, applicative logic and to add a set of type-shifting rules in the spirit of combinatorial logic. Two types categorial grammars using this alternative setup are combinatory categorial grammar (Steedman, 2001, CCG) and flexible Montague grammar (Hendriks, 1993). The choice of multiplicative intuitionistic linear logic also has some descriptive repercussions: as discussed in Section 3, the treatment of parasitic gapping and some treatments of anaphora and binding are incompatible with this choice and require a more powerful logic.

Example (1), called "right node raising" in the literature, shows one of the things which works correctly in the Lambek calculus: assuming assignments of $(np \setminus s) / np$ to the transitive verbs and np to "John" and "Peter", L derives "John loved" and "Peter hated" as expressions of type s / np. Though this works correctly, it crucially depends on the presence of associativity. For example, NL no longer allows us to derive sentence (1) unless we add a second lexical assignment $np \setminus (s / np)$ to the transitive verbs.

Examples like (2), and related examples, were first discussed in (Lambek, 1961, p. 167). The problem with this ungrammatical sentence is that "John read *Syntactic Structures* and Mindy liked" is a sentence missing a noun phrase (at its right edge) and can therefore combine with "which" given its standard Lambek calculus assignment $(n \setminus n) / (s / np)$. The same derivability pattern which helped us for sentence (1) leads to overgeneration for sentence (2). The ungrammaticality of this sentence is usually ascribed to "island constraints".

Examples (3) and (4) illustrate that the Lambek calculus has problems with quantifiers in medial position taking wide scope ("someone" in the *de re* reading of example (4)) and extraction from non-peripheral positions: sentence (3) is derivable without the sentence-final adverb "yesterday", again since "John read" is an expression of type s / np. However, the presence of "yesterday" blocks this derivation.

Another classic example is the treatment of Dutch verb clusters (Moortgat and Oehrle, 1994; Oehrle, 2011) (Sentence (5) above). This sentence exhibits the well-known crossed dependencies: "Henk" is the object of "zag" (saw), "Marie" the object of "helpen" (help) and "de nijlpaarden" (the hippopotami) the object of "voeren" (feed). Though Pullum and Gazdar (1982) show that such examples can be treated by context-free grammars, and hence by the Lambek calculus, the Dutch verb cluster analysis of mildly context-sensitive formalisms, which expresses the desired dependencies between objects and verbs is generally preferred.

Gapping (Hendriks, 1995) (Sentence (6), which has the same meaning as "John studies logic and Charles *studies* phonetics"), ellipsis (Sentence (7), which has the same meaning as "John left before Mary *left*") and comparative subdeletion (Sentence (8), which means "The number of girls Mary gave a book is greater than the number of boys *Mary gave* a record"), are some of the other interesting phenomena on the syntax-semantics interface which have been treated in type-logical grammars.

One general type of example where type-logical grammars do well are coordination and similar phenomena. The type-logical grammar treatment of coordination tells us we can conjoin two phrases whenever they can be assigned the same formula. Sentence (1) is an example, but also, in the analysis of Hendriks (1995), Sentence (6)[3]. The easiest way to implement this is to assign the second-order formula $\forall X.((X \setminus X) / X)$ to words like "and" and "but", though how to add second-order quantifiers while keeping type-logical grammars decidable is an open question.

A general type of extension to the Lambek calculus is exemplified by Moortgat's (1996a) $q(A, B, C)$ constructor, which has a rather simple left rule.

$$\frac{\Delta[A] \vdash B \quad \Gamma[C] \vdash D}{\Gamma[\Delta[q(A, B, C)]] \vdash D}\ qL$$

Its instantiation $q(np, s, s)$ gives a good account of quantifier scope, whereas reflexives can be assigned $q(np, np \setminus s, np \setminus s)$. In addition, it can be extended to treat comparative subdeletion and other phenomena as well (Sentence (8)) (Moortgat, 1996a). Carpenter's

[3]Though in the case of (Hendriks, 1995) and also in the similar case of (Morrill et al., 2011) this is a coordination of two formulas which are *almost* the same.

Formalism	Islands	RNR	Rel	q	$\wedge$	gap	Language	Complexity
NL	$+$	$-$	$-$	$-$	$+$	$-$	$=$ CFL	P
L	$-$	$+$	$-$	$-$	$+$	$-$	$=$ CFL	NP
D_{core}	$-$	$+$	$+$	$+$	$+$	$+$	$\supseteq \mathrm{MCFL}_{wn}$	NP
D_{full}	$+$	$+$	$+$	$+$	$+$	$+$	$\supsetneq$ MCFL	?
ACG_2	$-$	$-$	$-$	$-$	$-$	$-$	$=$ MCFL	NP
ACG	$-$	$-$	$+$	$(+)$	$-$	$-$	$\supseteq$ MCFL	NP
LG	$-$	$-$	$-$	$+$	$+$	$+$	$\supsetneq$ MCFL	NP
NL_λ	$+$	$-$	$+$	$+$	$+$	$+$	$\supsetneq$ MCFL	NP(?)
HTLG	$-$	$+$	$+$	$+$	$+$	$+$	$\supsetneq$ MCFL	NP
MILL1	$+$	$+$	$+$	$+$	$+$	$+$	$\supsetneq$ MCFL	NP
$\mathrm{NL}\Diamond_{\mathcal{R}}$	$+$	$+$	$+$	$+$	$+$	$+$	$=$ CSL	PSPACE

Table 1: Scorecard for the different extended Lambek calculi

(1998) textbook gives an introduction to semantics and type-logical grammar using mostly the Lambek calculus extended with the q operator. Now, semantically, the q operator behaves like a combination of an introduction rule and an elimination rule, so it seems the q connective may need to be decomposed and though we have given a left rule for q there is no easy way to formulate its *right* rule for q. Because of the simplicity and many applications of the q operator, these questions have led to many decompositions of q (Morrill, 1994; Moortgat, 1996b; Barker and Shan, 2014, to cite but a few)

The above list of sentences gives only a very partial picture of the descriptive work done in the various type-logical frameworks, but it provides a starting point for discussion.

Table 1 is my attempt to, as Lakatos (1978) suggests, keep score honestly. It is a table cross-referencing formalisms and phenomena, assigning a "+" if the formalism has a good treatment of the given empirical data and "−" if it does not.

The formalisms are, from top to bottom: the non-associative Lambek calculus, NL (Lambek, 1961), the associative Lambek calculus L (Lambek, 1958), the core, multiplicative fragment of the Displacement calculus, D_{core} (Morrill et al., 2011, this includes the multiplicative connectives, with the synthetic connectives restricted to $\hat{}$, $\triangleright^{-1}$ and $\triangleleft^{-1}$, when I use the term "D" or "the Displacement calculus" without further qualification in the rest of this paper, I mean this fragment), the full Displacement calculus, D_{full} (Morrill and Valentín, 2015, this volume). Abstract categorial grammars/lambda grammars, ACG and their second-order restriction ACG_2 (de Groote, 2001; de Groote and Pogodalla, 2004), the Lambek-Grishin calculus, LG (Bernardi and Moortgat, 2010), NL_λ (Barker and Shan, 2014), Hybrid Type-Logical Grammars, HTLG (Kubota and Levine, 2012, 2013), first-order linear logic, MILL1 (Moot and Piazza, 2001) and the multimodal Lambek calculus $\mathrm{NL}\Diamond_{\mathcal{R}}$ (Moortgat, 1996c, 1997).

The phenomena are, from left to right: does the formalism have a way, such a non-associativity, to encode island constraints? Does the formalism have a way to encode right-node raising (RNR)? Does the formalism have a way to handle medial extraction? Does the formalism have a way to simulate the q operator[4]? Does the formalism handle coordination ($\wedge$)[5]? Does the formalism treat gapping and related phenomena? What do we know about the language class of the given formalism (the classes are context-free

[4]A "(+)" means the formalism can only do so partially. For example ACG does not handle $q(A, B, C)$ for complex formulas, for example those used for reflexives (Moot, 2014b).

[5]Coordination for atomic formulas only is marked as "−", a "+" for coordination means: "the formula $\forall X.((X \setminus X) \,/\, X)$ for conjunction works correctly for all formulas X in our lexicon"

languages, well-nested multiple context-free languages, multiple context-free languages and context-sensitive languages)? And finally, what is the computational complexity of the formal system?

Our knowledge of the precise language classes is very partial: this is partly due to the difficulty of extending a proof like (Pentus, 1995) to extended Lambek calculi (Buszkowski, 1997), which means few upper bounds are known and in general few formal proofs of generated language classes exist and the table entries represent answers to questions like "are there grammars of the language $a^n b^n c^n$, and ww?". Formalisms which generate (at least) the well-nested multiple context-free languages can correctly treat phenomena such as the Dutch verb clusters discussed above.

Not listed in the table is parasitic gapping. The full Displacement calculus with contraction modalities is the only calculus among those listed which has a treatment of parasitic gapping, as exemplified by a noun such as.

(9) contract which John filed without reading.

In these cases, the object noun phrase bound by "which" has been extracted twice, once as the object of "filed" and once as the object of "reading".

Also absent from the table are treatments of anaphora rooted in logical syntax, such as those of (Jäger, 2001; Morrill and Valentín, 2014; Barker and Shan, 2014). Like parasitic gapping, such treatments require a modification of the hypothesis, discussed in Section 2, of a linear syntax-semantics interface and like parasitic gapping, they complicate the proof theory. I believe that for anaphora (and binding theory) the proper division of labour between syntax, syntax-semantics interface and semantics/pragmatics is still very much open for debate and that integrating the many interacting facts on these different levels into a single, coherent system will be a difficult but important task.

The computational complexity column lists the complexity of the parsing or universal recognition problem (fixed recognition is a property of formal languages and has little relation to parsing complexity). All results listed "NP" or "PSPACE" indicate NP- or PSPACE-completeness. The complexity of the full Displacement calculus is unknown, and may be undecidable. The complexity of NL$_\lambda$ is unknown; Barker and Shan (2014) show only decidability, but the close connection to HTLG — both employ a combination of a standard Lambek calculus (L and NL respectively) together with lambda-calculus like operations (exactly those of the lambda calculus for HTLG and operations very close to it in NL$_\lambda$) — makes it likely to be an NP-complete system (since it appears polynomial time verifiable whether or not an HTLG proof is also an NL$_\lambda$ proof[6]). NL can be parsed in polynomial time (Aarts and Trautwein, 1995; de Groote, 1999).

Though it is easy to criticise this table and talk about important missing phenomena or incorrectly assigned entries, I hope this will provide a starting point for discussion and provide an impetus for people to "improve their score" on the table. I would like to add that systems with a "low score", such as the Lambek calculus, are still worthwhile systems to use and study, as long as we are aware of their limitations for certain applications. My goal is to allow "people to do their own thing but only as long as they publicly admit what the score is between them and their rivals" (Lakatos, 1978, p. 100).

[6]Of course this presupposes first that NL$_\lambda$ proofs are a subset of the HTLG proofs, in the same sense that NL proofs are a subset of L proofs.

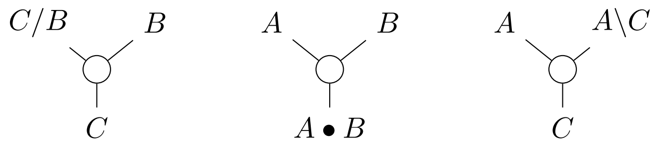

Figure 1: Binary branches and the connectives of NL

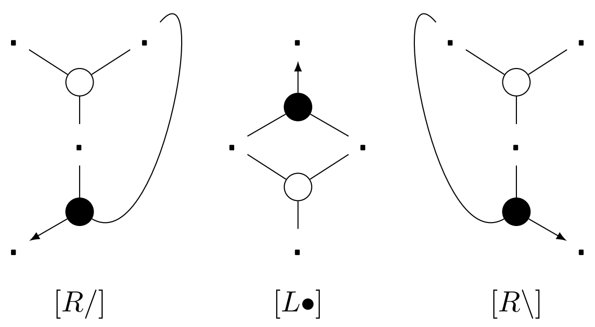

Figure 2: Contractions — Lambek connectives

4 Graph rewriting

The first meta-logic for creating bridges between the different syntactic derivations of extended Lambek calculi is a simple generalisation of the proof net calculus of Moot and Puite (2002). Its basic idea is that you only need to specify the *structures* of your logic (ie. the way your formulas are organised in the sequence, for example, as a list, a tree, or as a graph) and then you will automatically obtain the possible connectives and a correctness criterion based on graph rewriting and contractions in the style of Danos (1990) and close to the interaction nets of (Lafont, 1995).

For example, if we choose binary branching trees as structures, then we obtain three connectives depending on which of the three related nodes contains the main formula and these are just the familiar connectives of the non-associative Lambek calculus NL.

In addition to the constructors shown in Figure 1, each connective, by logical symmetry, induces a destructor[7] as well. These destructors have the property that an appropriately connected constructor/destructor pair contracts to a point. When it is impossible to contract a destructor, the graph does not correspond to a provable statement; provable statements contract to a (constructor) tree.

The contractions for NL are shown in Figure 2. Each destructor is drawn in black and has a single outgoing arrow (this arrow points to the main formula of the link).

We can contract only when the branches of a constructor and destructor are connected by the two branches without the arrow. For the $[L\bullet]$ contraction, this means we connect the two left branches to each other and similarly for the two right branches. The contractions for the implications look a bit odd when we draw them on the plane and preserve left/right and top/bottom of the links. The $[R/]$ contraction connects a constructor and destructor

[7]For the binary connectives of linear logic, the constructors are tensor links and the destructors are par links. In the terminology of focusing proof search (Andreoli, 1992), we would talk about synchronous or non-invertible rules instead of constructors and asynchronous or invertible rules instead of destructors.

top-to-bottom while connecting the two right branches (the left branch has the arrow pointing towards it). Once we have connected top and bottom, we are forced to bend one of the right branches (alternatively, we could write the graph on a cylinder). The idea behind the $[R/]$ contraction is that it withdraws a hypothesis occurring as the immediate right daughter of the root of the tree, just like the sequent calculus or natural deduction rule for NL would do. The bent link can be seen as a graphical representation of the coindexing of the hypothesis with the rule application for the $/I$ rule. By simple pattern-matching with Figure 1 we can see that these simple instances of the $[R/]$ contraction corresponds to proofs of $A \,/\, B \vdash A \,/\, B$, $A \vdash (A \bullet B) \,/\, B$ and $A \vdash C \,/\, (A \setminus C)$.

The important idea of take away from this is that we specify our structures (in the case of NL, these were simple binary branching trees), and that from this we automatically obtained three connectives, where each of the three possible "main edges" induced a corresponding contraction.

Moving from NL to the multimodal Lambek calculus NL$\Diamond_{\mathcal{R}}$ means changing our structures from unlabelled binary trees to labelled binary-unary trees, that is labelled trees where both binary and unary branches are allowed. The unary branches corresponds to the connectives $\Box$ and $\Diamond$, and the labels, which we draw inside the central circle of a branch, correspond to mode information. The contractions have the additional condition that the modes of the constructor and destructor must be the same.

The final component of the calculus is the set of structural rules, which are rewrites from one subtree of the graph to another. The unary branches and the mode information control which structural rules can apply, so we can have a non-associative base logic yet still handle right-node-raising and quantifier scope (Kurtonina and Moortgat, 1997; Moortgat, 1996b). Another way to see the structural rules it that they specify a partial order on our structures.

For reasons of space, this introduction to the graph rewriting approach to proof nets is a bit impressionistic, for much more detail and the fundamental results, the reader is referred to (Moot and Puite, 2002; Moot and Retoré, 2012). Though created as a proof net calculus for multimodal categorial grammars, it can easily be generalised to handle the Lambek-Grishin calculus, which adds up-down symmetric structures to NL (see the left hand side of Figure 3), and the Lambek calculus with Galois connections (Areces et al., 2001), which adds negation-like polarity changes to the calculus (see the middle of Figure 3). Other calculi also have a presentation which is equivalent to a multimodal calculus, so we can use this proof net calculus also for Valentín's (2014) multimodal version of the Displacement calculus and for NL$_{CL}$, a multimodal version of NL$_{\lambda}$ (Barker and Shan, 2014, Chapter 17), which adds zero-ary connectives ("constants") to the multimodal calculus. All these proof net calculi are simple instances of the general methodology: specify the structures (and possibly a partial order on them) and obtain the connectives and the contractions for a proof net calculus (Moot, 2007).

In the next sections, I illustrate with the help of two examples how this proof net calculus allows us to compare different systems.

4.1 Lambek-Grishin and Galois connections

First, we can see that the non-associative Lambek calculus (NL) extended with Galois connectives has the Lambek-Grishin calculus as a special case. As Figure 3 shows, LG extends NL with up-down symmetric connectives whereas the Galois connectives give a negation-like polarity change operation. In particular, as shown in the center of Figure 3, this allows us to simulate the Grishin connectives: a combination of an NL constructor with a Galois connective as shown in the figure forms a subgraph which to the outside "looks

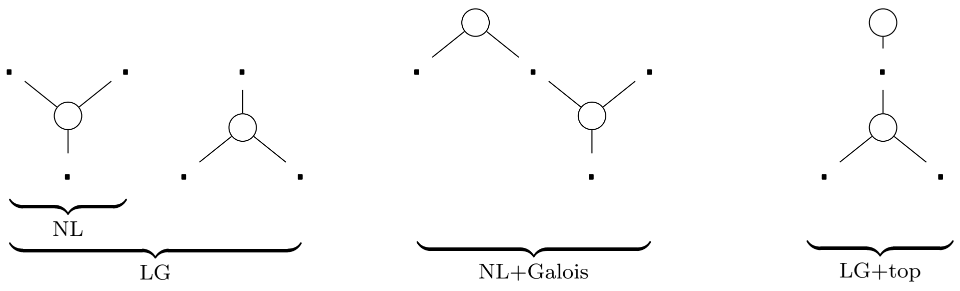

Figure 3: NL, LG and Galois connections

the same", up until the cyclic order of the external vertices, as the Grishin connective (there is an alternative solution, which is left-right symmetric to the one shown).

This mapping goes only in one direction, however. The translation back to LG is partial: though the images of translations of LG graphs are sent back to the original graph, a graph consisting of a single Galois connective cannot be translated to LG, since only the "unit" displayed in the center of the figure is a meaningful candidate for such a translation.

However, suppose there is something interesting in NL+Galois and we are interested in translating this analysis to LG. The graphs immediately suggest the solution shown on the right of Figure 3: a combination of a Grishin connective with a 0-ary "top" connective can function as a Galois connective. Like before the two graphs look the same to the outside world: both have no premisses and two conclusions in the same order (a 0-ary connective is not the same as an identity element: if we want such a connective to function as an identity element for a binary connective we need to add the appropriate structural rules, an example of such a structural rule can be found on the left of Figure 6 below).

To complete the picture, we have the following relations between NL, LG and the Galois/top connectives ($A \subset B$ means here that we can translate structures of logic A into structures of logic B in such a way that logic B is a conservative extension of logic A).

$$\text{NL} \subset \text{LG} \subset \text{NL+Galois} \subset \text{LG+top} = \text{NL+Galois+top}$$

If we want a fully up-down symmetrical system, we need to complete the system with dual-Galois connectives and a bottom connective in the obvious way.

4.2 Multimodal versions of q

As a second illustration, Moortgat (1996b) and Barker and Shan (2014) both provide an implementation of the in situ binder $q(A, B, C)$, shown in Figure 4 and Figure 5 on the right and center resp. left and center (both have an additional "start" rule shown in Figure 6, discussed below). For the rules of Figures 4 and 5 the two substructures displayed in gray give us a mapping from one system to the other. This highlights both similarities and differences between the two approaches: the Barker and Shan implementation, which they call NL_{CL}, works unchanged when multiple in situ binders "overlap", as used for their analysis of the semantics of "*same*", whereas Moortgat's approach works correctly even in the presence of associativity. Both properties are a consequence of the unary branches and their position in Moortgat's calculus: the unary branches block associativity, but also

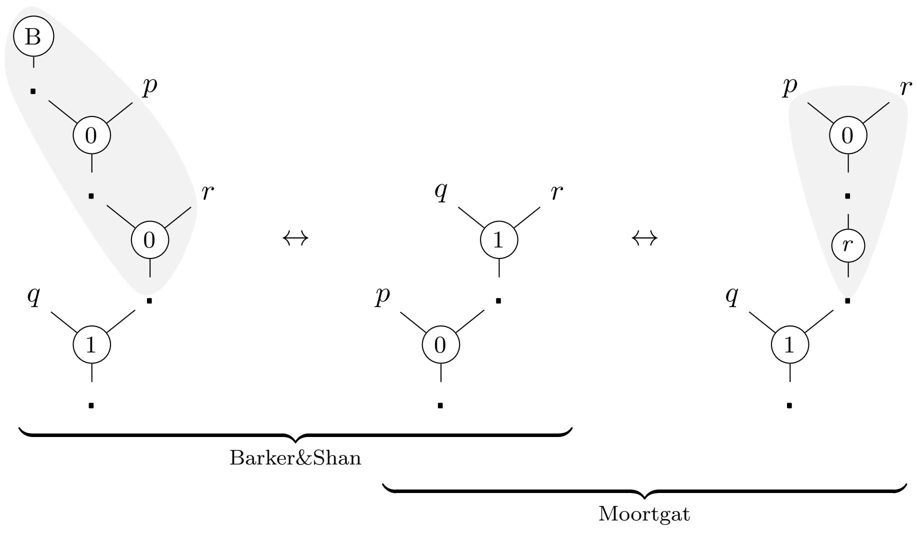

Figure 4: Mixed commutativity in (Barker and Shan, 2014) compared to (Moortgat, 1996b)

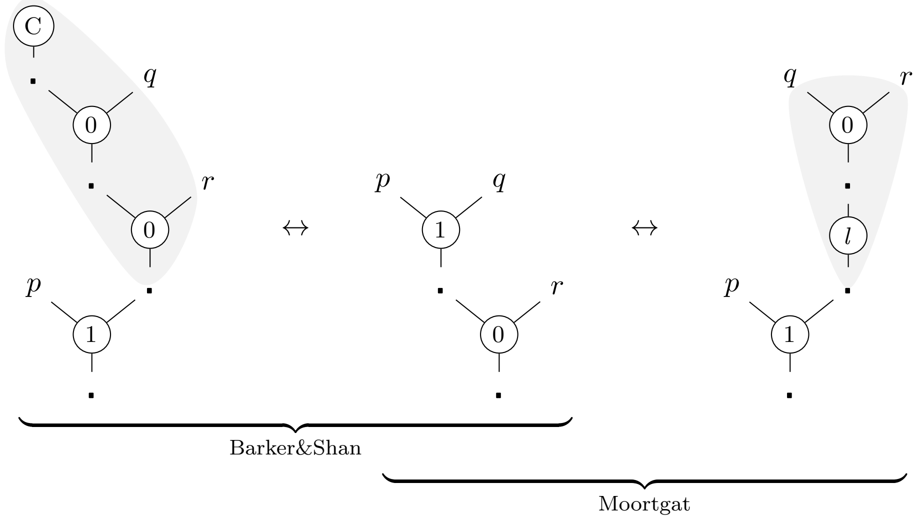

Figure 5: Mixed associativity in (Barker and Shan, 2014) compared to (Moortgat, 1996b)

prevent the rules from operating in the direction from center to right when there is a unary branch between the two constructors.

In Moortgat's original system, the unary branches leave a "trail" telling us at each binary branch whether to go left or right. In the Barker and Shan system, there is a similar trail but stored at a deeper position. Barker and Shan's system breaks down when we add associativity to the system — though the addition of modally controlled associativity seems delicate even in Moortgat's system and needs to be carefully investigated.

Finally, there are the start rules of Figure 6. For the Barker and Shan system the I connective is simply an right identity element for the binary mode 1 (the structural rule looks strange compared to the ones we've seen before, but it is of the same shape: we replace a subtree with leaf p, binary mode 1 and 0-ary mode I by the leaf p, which is a

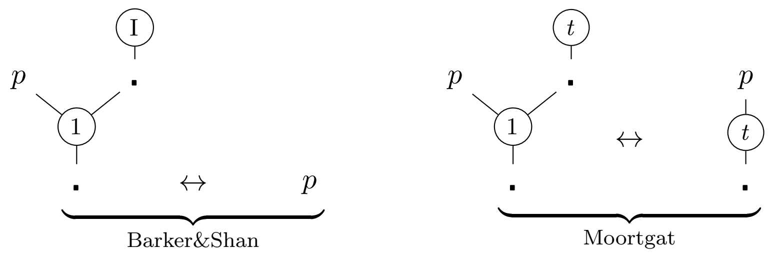

Figure 6: Start rules of (Barker and Shan, 2014) compared to (Moortgat, 1996b)

subtree with the same leaves). In Moortgat's version, we replace a unary branch t by a binary branch with a right identity element t (t occurs both as a unary branch and as a 0-ary branch in this rule). This formulation has the advantage that we can lexically control the application of the rule (the current form of these rules is at least partially motivated from the point of view of facilitating automated deduction for the system). For the Barker and Shan implementation, we need to be careful when applying the rule from right-to-left, since naive application of this rule can lead us into an infinite loop.

To intertranslate the start rules, we can either use the fact that we can eliminate "useless" applications of the I rule from NL_{CL} — we know from the embedding results of Sections 17.5 and 17.7 and Definition 304 on page 172 of (Barker and Shan, 2014) that we can, without loss of generality, restrict NL_{CL} structures containing I only at the end of a path of B and C nodes — or import Moortgat's analysis directly into NL_{CL}. In either case, it is then a simple translation between 0-ary I and 0-ary t.

Looking at Figures 4 and 5 also shows us exactly when we can translate NL_{CL} structures with BCI connectives to Moortgat's calculus: only when B and C nodes occur as the right daughter of two consecutive $\circ_0$ branches, that is only when they occur as $(B\circ_0 X)\circ_0 Y$ or $(C \circ_0 X) \circ_0 Y$ (for some structures X and Y). But this property crucially fails when there are multiple binders, such as in the term below, which occurs in the analysis of "the same waiter served everyone" on page 166 of (Barker and Shan, 2014).

$$\mathit{same} \circ_1 ((C \circ_0 ((B \circ_0 \mathbf{B}) \circ_0 ((B \circ_0 \mathit{the}) \circ_0 ((C \circ_0 I) \circ_0 \mathit{waiter})))) \circ_0 ((B \circ_0 \mathit{served}) \circ_0 I))$$

In the above tree (presented as a flat term to preserve space) the B indicated in bold is not translated when we simply intertranslate the structures indicated in gray in Figures 4 and 5, since it occurs on a right branch. The translation produces the following term which still contains this B which is not a part of Moortgat's calculus (the circumfix $\langle . \rangle^i$ denotes a unary branch with label i).

$$\mathit{same} \circ_1 \langle \langle \mathbf{B} \circ_0 \langle \mathit{the} \circ_0 \langle t \circ_0 \mathit{waiter} \rangle^l \rangle^r \rangle^r \circ_0 \langle \mathit{served} \circ_0 t \rangle^r \rangle^l$$

In the Barker and Shan analysis, this B is part of the second path; after infixation of "same" using the structural rules, we obtain the following term, with the same B shown in bold.

$$(\mathbf{B} \circ_0 (\mathit{the} \circ_0 (\mathit{same} \circ_0 \mathit{waiter}))) \circ_0 ((B \circ_0 \mathit{served}) \circ_0 I)$$

This second term translates unproblematically to the following term.

$$\langle (\mathit{the} \circ_0 (\mathit{same} \circ_0 \mathit{waiter})) \circ_0 \langle \mathit{served} \circ_0 t \rangle^r \rangle^r$$

Though this analysis does not immediately show us the best way to extend Moortgat's calculus to handle multiple binders, it does give us an indication of where to look: we want to extend the translation in such a way that B terms can no longer appear and which allows us to rewrite our new terms, no longer containing B, to the term above. It is not hard to add modes and structural rules to allow exactly this, but I'll leave more elegant solutions to future research.

4.3 Graph rewriting, graph grammars and modal logic

I have briefly talked about how graph rewriting can be a tool for providing mappings from one type-logical grammar to another. There are some other connections I briefly want to mention.

Since most type-logical grammars have models in modal logic, can we use some of the rich set of tools of modal logic to discover relations between different formalisms? For example by using bisimulations between the model-theoretic interpretations of two calculi (Blackburn et al., 2001). This would be a model-theoretic counterpart to the proof-theoretic approach of the current paper. Also, since the modern view of modal logic sees modal logic as (decidable) fragments of first-order logic this ties in nicely with the first-order perspective of the next section[8].

A second point is that graph rewriting has been extensively studied in computer science. However, these tend to be nicer classes of graphs than those presented here. Is there a connection between some context-free graph grammars (Engelfriet, 1997) and the proof nets studied here or do we, as seems likely, need richer classes? If so, which ones and what are their properties? Some preliminary investigations into this topic can be found in (Moot, 2008).

5 First-order linear logic

The second meta-logic is first-order (multiplicative, intuitionistic) linear logic. Though the idea of adding first-order quantifiers to type-logical grammars is very old (Morrill, 1994, Section 6.2), the important shift in Moot and Piazza (2001) was that the first-order quantifiers could also be used for word order. So instead of adding first-order quantifiers to one of the other variants of type-logical grammar, we can add first-order quantifiers to multiplicative linear logic and embed the Lambek calculus. Several of the known problems of the Lambek calculus have a simple solution in first-order linear logic. First-order linear logic also has a very nice, clean proof net calculus (Girard, 1991).

5.1 Proof search in first-order linear logic

I will give a very brief introduction to first-order linear logic from the point of view of proof search and parsing. Figure 7 presents the logical links for first-order linear logic proof nets. To try and prove a sequent $A_1, \ldots, A_n \vdash C$ we decompose the formulas according to the links of the figure, starting with

$$\overset{-}{A_1} \ldots \overset{-}{A_n} \overset{+}{C}$$

and continuing the unfolding until we reach the atomic subformulas. The links essentially produce a subformula tree with the additional marking of the polarity of each subformula and with some dotted lines. There are two types of dotted lines: those which pair two branches (for the positive $\multimap$ and negative $\otimes$ link) and those which are labelled with the

[8]I thank one of the anonymous referees for pointing out the connection to modal logic.

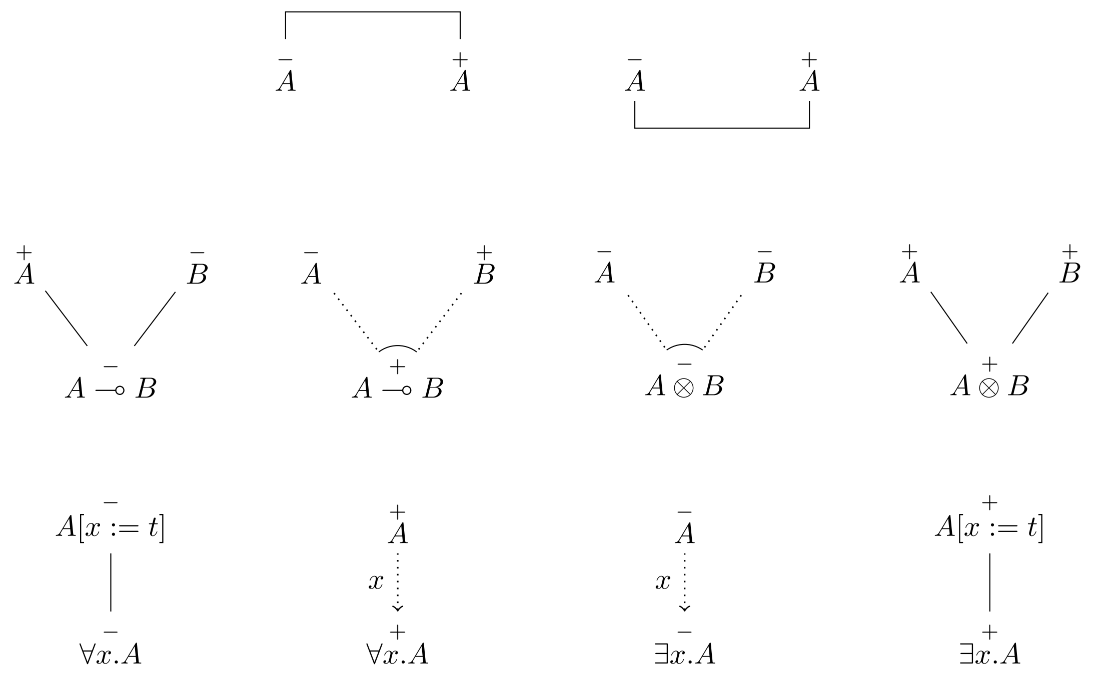

Figure 7: Logical links for MILL1 proof structures

eigenvariable of a quantifier (for the positive $\forall$ and the negative $\exists$ link). We use the standard convention that each quantifier in a sequent has a different eigenvariable.

After this unfolding step, we connect atomic formulas of opposite polarity using the axiom link (Figure 7, top left). When all atomic formulas have been connected this way, the resulting structure is called a proof structure. As an example, the sequent $\forall x.a(x) \multimap b \vdash \exists y.[a(y) \multimap b]$ has the proof structure shown in Figure 8.

In practice, we do not choose a term for the negative $\forall$ or for the positive $\exists$ link during formula unfolding but we rather use meta-variables for unfolding and unification during the axiom link connections. So the axiom link does not connect identical formulas but rather unifiable formulas and performs this unification. This is a rather standard theorem-proving strategy and has the result that we can read off the most general term for each negative $\forall$ and positive $\exists$ rule in our proof net. Some care must be taken when we repeatedly unify the positive and negative atomic subformulas of $\forall x.a(x) \multimap a(f(x,x))$, where the size of the term argument grows exponentially in the number of occurrences of the given formula $(a(x), a(f(x,x)), a(f(f(x,x),f(x,x))), a(f(f(f(x,x),f(x,x)),f(f(x,x),f(x,x)))), \ldots)$. Even in these cases, we can ensure linear time unification by adopting a sharing strategy (Martelli and Montanari, 1982; Patterson and Wegman, 1978). In addition, for the typical uses of first-order linear logic which interest us here, each quantifier binds at most two occurrences of its eigenvariable (Moot, 2014a), so we might even decide to exclude the case above, where the quantifier $\forall x$ binds three occurrences.

Returning to the example proof structure of Figure 8, the given sequent is underivable in linear logic and, in general, proof structures need not correspond to proofs. Proof structures which correspond to proofs are called *proof nets* and we can distinguish them from other proof structures using properties of the graph. Girard (1991) gives a switching criterion whereas Moot (2014a) presents a contraction criterion for first-order linear logic in the style of Danos (1990).

The contractions for first-order linear logic are shown in Figure 9. As a first step, we

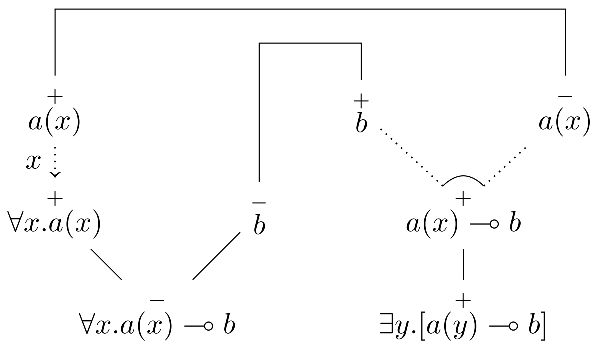
$\overset{+}{a(x)}$
x
$\overset{+}{\forall x.a(x)}$
$\overset{-}{b}$
$\overset{-}{\forall x.a(x) \multimap b}$
$\overset{+}{b}$
$\overset{-}{a(x)}$
$\overset{+}{a(x) \multimap b}$
$\overset{+}{\exists y.[a(y) \multimap b]}$

Figure 8: Proof structure which is not a proof net

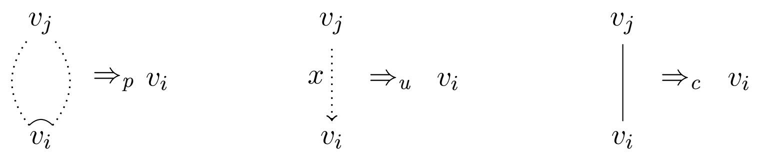

Figure 9: Contractions for first-order linear logic. Conditions: $v_i \neq v_j$ and, for the u contraction, all free occurrences of x are at v_j.

forget about all formulas in the proof structure keeping track only of the free variables at each vertex in the graph. For Figure 8 this produces the graph shown on the left hand side of Figure 10. Then we progressively shrink the proof structure using the contractions shown in the figure. Each contraction removes an edge (a linked pair of edges in the case of the p contraction) and identifies the two nodes which were connected by it. A contraction can only be performed on two distinct vertices, that is, we are not allowed to eliminate self-loops. The free variables of the result vertex are the union of the sets of free variables of the two input vertices (in the case of the u contraction, we can remove the x variable, since it has become redundant). A proof structure is a proof net if and only if it contracts to a single point using the contractions of Figure 9.

As an example, Figure 10 shows the contractions performed on the proof structure of Figure 8, with the initial structure on the left and the structure after all c contractions on the right. The arrow and eigenvariable of the $\forall$ link and the connection between the

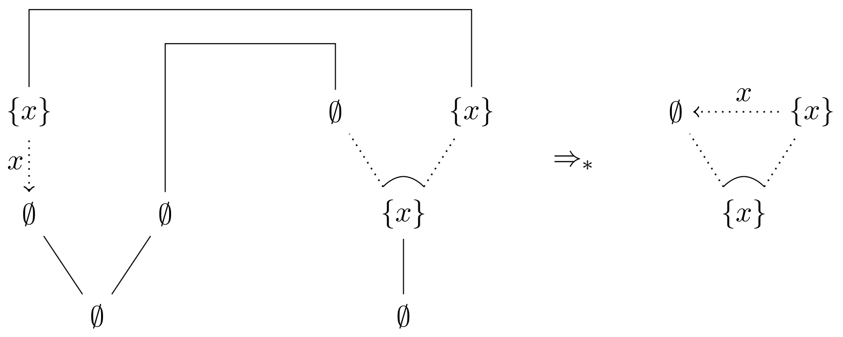

Figure 10: Contractions for the proof structure of Figure 8

two other dotted links ensure the notation is unambiguous. The displayed graph is not a single vertex but it cannot be further contracted: the universal contraction u cannot apply since the variable x occurs at the bottom vertex instead of only at the right vertex as required for the contraction and the contraction p cannot apply until its two branches end at the same vertex. Since the contractions of Figure 9 are confluent[9], any graph which is not further contractible but is not a unique vertex, such as the one shown at the right of Figure 10, suffices to show that the given sequent is underivable.

Summarising, parsing/proof search in first-order linear logic operates as follows.

1. For each word in the sentence, we find a first-order formula in the lexicon.
2. We unfold a sequent using the rules of Figure 7.
3. We connect atomic formulas of opposite polarity, unifying variables.
4. We contract the resulting proof structure to a single vertex.

Combinatorially, the complex steps are step 1 (in the case of high lexical ambiguity) and step 3 where we connect the atomic formulas. For an actual implementation, such as (Moot, 2015), it is therefore desirable to contract early — thereby keeping a compact representation of the current state of the proof — and develop ways of detecting "doomed" configurations, that is graphs which can never be contracted to a single vertex, no matter how we continue the construction of our proof structure. Examples of such configurations are connections between a node and its ancestor with a path of dotted links (this corresponds to a cycle in the proof structure and though we can validly reduce the size of this cycle, such a configuration will, at best, end up producing a self-loop) or isolated vertices (an isolated vertex is a vertex which is not connected to the rest of the graph but which also doesn't have any unlinked atomic formulas; unless it is the last vertex of the graph, such a vertex corresponds to a disconnected proof structure). Combining early failure with a smart backtracking strategy for selecting which atomic formulas to unify (Knuth, 2000) produces an effective algorithm, though I suspect many improvements are still possible.

5.2 First-order linear logic as a grammar formalism

Whereas Moot and Piazza (2001) show that the Lambek calculus has a natural translation into first-order linear logic, in (Moot, 2014a,b), this idea is further developed and translations are given which show that Displacement grammars, Hybrid Type-Logical Grammars and lambda grammars are all natural fragments of first-order linear logic, providing much simpler proof-theoretic foundations for these calculi. In addition, the analyses proposed in these different frameworks agree to a large extent upon translation into first-order linear logic. The basic idea of the translation into first-order linear logic is that formulas are assigned pairs of string positions. For example, an atomic Lambek calculus formula np becomes an atomic first-order linear logic formula $np(0,1)$ when it spans the string from position 0 to 1. Similarly, the Lambek calculus formula $np \setminus s$ spanning the string from 1 to 2 becomes the formula $\forall x.np(x,1) \multimap s(x,2)$, indicating it is looking for a noun phrase to its right to form a sentence. Instantiating x to 0, we can combine it with $np(0,1)$ to derive $s(0,2)$, a sentence spanning string positions 0 to 2. The translations for Hybrid

[9]As discussed in (Moot, 2002), this is not true for the contractions of Section 4 which require us to explore the entire search space to show underivability. The culprits in that case are the unary contractions and the structural rules.

Type-Logical Grammars and the Displacement calculus, though based on essentially the same idea, are a bit more involved and will not be repeated here.

Compared to the graph rewriting of Section 4, this setup has the advantage that instead of comparing *structures* and giving mappings from structures to structures — which has the difficulty that we need to show these structures behave the same in *all* contexts — we can translate to first-order linear logic and compare *formulas*. This is much easier and more immediate: we translate to first-order linear logic and compare the formulas we obtain. So we can see, for example, that the treatment of gapping in Hybrid Type-Logical Grammars and the one of the Displacement calculus, in spite of being formulated using very different logical primitives, are actually *equivalent* upon translation into first-order linear logic (in the sense that the translated formulas are interderivable). There is no need to specify a translation of the primitives of one calculus into another nor to provide a mapping from proofs to proofs.

As a simple example, the lambda grammar/ACG lexical entry $np \multimap s$ with prosodic term $\lambda S.(S + \mathit{sleeps})$ becomes, according to the translation into first-order linear logic, the formula $\forall x.np(x,1) \multimap s(x,2)$, just like the Lambek calculus (and Displacement calculus) formula $np \setminus s$. Even though these translations follow rather different paths, they end up at the same destination and it is this agreement on many of the "basic" lexical entries which forms the basis of the comparison of formalisms using first-order linear logic.

5.3 Relative pronouns

As a more interesting example of this way of comparing formulas, here are five different first-order linear logic formulas expressing extraction. These formulas would be assigned to a relativiser such as "which" occurring at position 3-4.

(1) $\forall x_0.[(\forall x_1.[np(x_1,x_1)] \multimap s(4,x_0)) \multimap \forall x_2.[n(x_2,3) \multimap n(x_2,x_0)]]$

(2) $\forall x_0.[\exists x_1.[np(x_1,x_1) \multimap s(4,x_0)] \multimap \forall x_2.[n(x_2,3) \multimap n(x_2,x_0)]]$ D

(3) $\forall x_0 \forall x_1 \forall x_2.[((np(x_1,x_1) \multimap s(4,x_0)) \multimap (n(x_2,3) \multimap n(x_2,x_0))]$ λ-grammar

(4) $\forall x_0.[\forall x_1.[np(x_1,4) \multimap s(x_1,x_0)] \multimap \forall x_2.[n(x_2,3) \multimap n(x_2,x_0)]]$ L : $(n\backslash n)/(np\backslash s)$

(5) $\forall x_0.[\forall x_1.[np(x_0,x_1) \multimap s(4,x_1)] \multimap \forall x_2.[n(x_2,3) \multimap n(x_2,x_0)]]$ L : $(n\backslash n)/(s/np)$

The first three formulas, though with slightly different scopes for the quantifiers, intuitively mean that a relative pronoun spanning positions 3-4 is looking to its left for a noun n (a noun spanning positions x_2-3 for some x_2 of our choice) and to its right for a sentence s, which itself is missing a noun phrase anywhere (where this sentence spans positions 4-x_0 for some x_0 of our choice, the relation between the position of the np and this sentence is not specified, though the proof theory will ensure this np will occur "inside" the sentence). The result will be a noun from position x_2, the start of the n argument, to x_0, the end of the s argument.

The first formula is a possibility which I have not seen before. Formula 2 is the formula from (Moot and Piazza, 2001) as well as the translation into first-order linear logic of the extraction formula for the Displacement calculus (D). Formula 3 is the translation of the lambda grammar lexical entry proposed by Muskens (2001). Finally, formulas 4 and 5 are the translations of the two Lambek calculus formulas for peripheral extraction. These formulas are related as follows (where a directed path between the two formulas denotes derivability of the target from the source).

$$\begin{tikzcd} & & & 4 \\ 1 \arrow[r] & 2 \arrow[r, shift left] & 3 \arrow[l, shift left] \arrow[ur] \arrow[dr] & \\ & & & 5 \end{tikzcd}$$

So the formulas of the Displacement calculus and the one proposed directly for first-order linear logic are identical (formula 2). This formula is equivalent to formula 3 proposed for λ-grammars; though we cannot always transform a linear logic formula into an equivalent prenex normal form (Lincoln and Shankar, 1994), formula 2 does allow such a form which is formula 3. When we look at lambda grammars in isolation, we cannot even directly ask the question about the relation to Lambek calculus formulas, though here it is clear that formulas 1 to 3 all have the Lambek calculus formulas 4 and 5 as special cases. The new formula 1 is the most general formula, but it is unclear whether or not there is any useful (or harmful!) difference in behaviour between this formula and the formulas corresponding to those use in the Displacement calculus and lambda grammars.

This brings up an important question: since, in (classical) first-order logic, formula 1 is equivalent to formulas 2 and 3 maybe first-order *linear* logic is too fine-grained a tool and the suitable notions of equivalence are better formulated directly in first-order logic. Is the difference between classical first-order equivalence and linear first-order equivalence important, and if, so which is the more suitable notion in the current context?

5.4 Adverbs, higher-order formulas and lambda grammars

The first-order linear logic perspective also clarifies the limitations of abstract categorial grammars/lambda grammars. For adverbs, for example, we are looking for a lexical entry which functions at least as well as the Lambek calculus formula $(np \setminus s) / (np \setminus s)$. However, as shown in (Moot, 2014b), we can simply enumerate all possible ACG lexical entries l, compute their translation into first-order linear logic, compute the translation of $(np \setminus s) / (np \setminus s)$ into first-order linear logic and compare. Keeping only the plausible lexical entries (that is, those which generate the right semantics and right word order) leaves us with three possibilities, which are shown as items 7 to 9 below, together with the translation of $(np \setminus s) / (np \setminus s)$ as item 6 (note the narrow scope of $\forall x_1$ in this translation). The adverb is assumed to span positions 1-2.

$$\forall x_0 \forall x_2 . [\forall x_1 . [np(x_1, 2) \multimap s(x_1, x_2)] \multimap (np(x_0, 1) \multimap s(x_0, x_2))] \tag{6}$$

$$\forall x_0 \forall x_1 \forall x_2 . [(np(x_1, x_1) \multimap s(2, x_2)) \multimap (np(x_0, 1) \multimap s(x_0, x_2))] \tag{7}$$

$$\forall x_0 \forall x_1 \forall x_2 . [(np(1, 2) \multimap s(x_1, x_2)) \multimap (np(x_0, x_1) \multimap s(x_0, x_2))] \tag{8}$$

$$\forall x_0 \forall x_1 \forall x_2 . [(np(x_1, 2) \multimap s(x_0, x_2)) \multimap (np(x_1, 1) \multimap s(x_0, x_2))] \tag{9}$$

The translations of ACG lexical entries are always formulas with only universal quantifiers and in prenex normal form[10]. It is easy to verify that all of items 7 to 9 are strictly more general than the translation of $(np \setminus s) / (np \setminus s)$, shown as item 6.

However, where in the case of relative pronouns, a more general formula turned out to be a benefit, in the case of adverbs, it turns out to be a source of overgeneration. For example, item 7, the adverb lexical entry most commonly used in the ACG literature,

[10]The term Skolem normal form is often used for a prenex normal form with universal quantifiers, but I don't use it here since it suggests that existential quantifiers have been replaced by universally quantified Skolem terms and 1) there are no terms in the translations of ACG formulas 2) Skolemization is unsound in first-order linear logic. (Lincoln and Shankar, 1994)

predicts that an adverb selects to its right, a sentence missing a noun phrase *anywhere*. In other words, the lexical entry for adverbs is modelled after the lexical entry for relative pronouns and therefore follows a "medial extraction" analysis, whereas items 8 and 9 predict a type of quantifying-in behaviour: item 8 is modelled after the type assigned to a generalised quantifier but with an extra *np* argument; it takes as its argument a sentences missing a noun phrase at the position of the adverb (just like a generalised quantifier takes a sentence missing an *np* at the position of the quantifier as its argument), making the odd prediction that adverbs occur at the same place as noun phrases. Formulas 7 and 8 therefore predict Sentences (10) and (11), along with many other strange possibilities, are correct.

(10) John deliberately Mary hit.

(11) Mary the friend of deliberately left.

Other higher-order Lambek calculus formulas have similar problems when we try to translate them into ACG. For example, the word "and" when used for the coordination of transitive verbs has 3024 possible translations, with 420 generating the correct surface structure and 148 having, in addition, the correct semantics as a possible reading. However, these many possibilities all follow the same pattern we have seen above for adverbs: they use a combination of extraction-like and quantifying-in constructions and therefore overgenerate.

5.5 Reflexives in D and in first-order linear logic

There is a minor difference between the Displacement calculus and its translation into first-order linear logic. The object reflexive of Morrill et al. (2011).

$$((vp \uparrow_{>} np) \uparrow_{<} np) \downarrow_{<} (vp \uparrow_{>} np)$$

is translated as follows, when it occurs at string positions 3-4 (the *vp* subformulas have been left untranslated).

$$\forall x_0, x_1, x_2, x_5 [np(3,4) \multimap np(x_1,x_2) \multimap \|vp\|^{x_0,x_5}] \multimap np(x_1,x_2) \multimap \|vp\|^{x_0,x_5}$$

The first-order linear logic formula more or less states that it transforms a ditransitive verb (with one of its objects occurring at position 3-4) into a transitive verb. However, the original D formula specifies something more, namely that the other *np* argument, $np(x_1, x_2)$ occurs before this first *np*, in other words that $x_2 \leq 3$. It is this property which is used to ensure the following two grammaticality judgments.

(12) Mary talked to John about himself.

(13) $*$ Mary talked about himself to John.

For formulas of the form $(A \multimap B \multimap C) \multimap D$, the Displacement calculus can encode the precedence between the A and B subformulas, whereas its translation in first-order linear logic cannot, in can only unify positions but has no mechanism for making statements like $x_2 \leq 3$[11].

[11]It would be possible to enrich first-order linear logic with order constraints on the variables, specifying things like $x_1 \leq x_2$. Though such an extension would be simple to implement, especially in a constraint-programming context, it seems to be rather powerful and it would complicate the proof theory, since such constraints would in all likelihood need to be non-linear.

Though I think the application of this property to reflexives is a clever solution, I am unsure if its unavailability in first-order linear logic is truly a handicap. The Displacement calculus refers to its "separators" by number in the linear order, whereas the lambda calculus-like formalisms give their "points for future insertion" a name (ie. a lambda calculus variable). Exploiting the core mechanism of the Displacement calculus to enforce constraints on linear order is a delicate undertaking since it risks interfering with the many other operations depending on these separators. For the treatment of Dutch verb clusters, Morrill et al. (2011, p. 28) are already aware that they sometimes need to refer to separators by mechanisms other than linear order.

5.6 Open questions

One important question must remain unanswered here: can we give (partial) translations from the Displacement calculus to Hybrid Type-Logical Grammars, or vice versa? We can, of course, translate into first-order linear logic and see if can translate it back to the other system (Moot, 2015, uses such a strategy, but to translate first-order linear logic proofs back to proofs in the "source language"). However, such a strategy depends on the exact formula obtained, for example the Formulas 2 and 3 on page 16 are equivalent and therefore this elementary case would already need some additional mechanism to cope with equivalence. A direct translation might side-step this problem and help crystallise the differences between the systems even more clearly.

There is another open question, which was already briefly alluded to when discussing unification. The variables and quantifiers of the formulas which interest us follow specific patterns. For example, each quantifier binds two occurrences of its eigenvariable (this is true for position variables such as those we have seen here, for agreement variables, such as case, grammatical gender etc., we may allow quantifiers which bind only one occurrence of their eigenvariable). Position variables further have one positive and one negative occurrence (where the switch from left position to right position is treated as a polarity change). Are there further patterns to discover here?

6 Towards convergence

Figure 6 gives a slightly simplified picture of the main logical calculi and their relations. The two meta-logics are not exclusive: the Displacement calculus can be embedded both in a graph rewriting calculus (as shown by (Valentín, 2014)) and in first-order linear logic (Moot, 2014a). Similarly, NL_λ seems to have a representation in both systems (Barker and Shan, 2014; Moot, 2014a,b), though the full formal details of this possibility still need to be worked out. However, the connection between a multimodal setup and a lambda calculus-like setup explored by Barker and Shan (2014, Chapter 17) seems an important step towards convergence of the meta-logics discussed in this paper.

It remains an open question whether there is a natural way to guarantee convergence of the two logics: what restriction on graph rewriting for proof nets would ensure that the calculus can be translated in first-order linear logic?

Similarly, it is unclear at the moment whether there is a connection between the Lambek-Grishin calculus (LG) and first-order linear logic. The multi-conclusioned nature of the Lambek-Grishin calculus makes it hard to compare it directly to its intuitionistic peers.

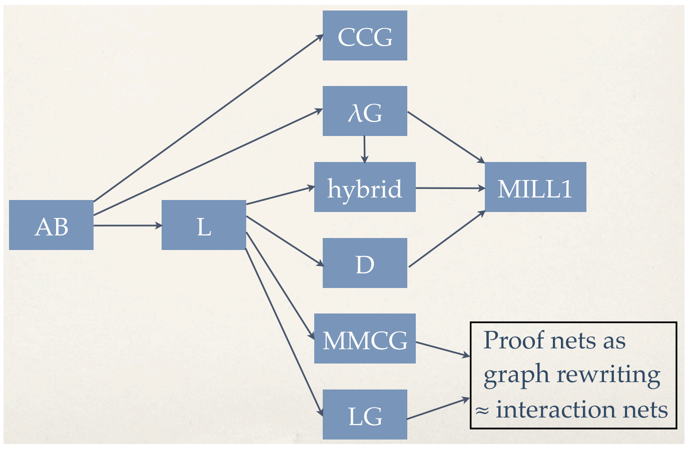

Figure 11: A (slightly simplified) representation of principal extensions and variants of the Lambek calculus (L). The Ajdukiewicz/Bar-Hillel calculus (AB), Combinatory Categorial Grammars (CCG), lambda grammars (λG), Hybrid Type-Logical Grammars (hybrid), the Displacement calculus (D), multimodal categorial grammars (MMCG) and the Lambek-Grishin (LG) calculus and the two meta-logics: first-order linear logic (MILL1) and the proof nets of Moot & Puite

7 Conclusions

In this paper I have given two tools which I hope will create bridges between the many formalisms in the family of type-logical grammars. These tools are not only theoretical tools, but they are also the basis of two implementations (Moot et al., 2015; Moot, 2015), a theorem prover based on (for the moment only the intuitionistic part of) the graph rewriting calculus and a theorem prover for first-order linear logic, which can output proofs for the Displacement calculus and Hybrid Type-Logical Grammars by translation.

I believe open discussion of the benefits and disadvantages of one system over another as well as a growing body of linguistic data which can find a satisfactory account in type-logical grammars will be an important factor in ensuring that our community stays vibrant and healthy.

Acknowledgements

I would like to thank Chris Barker, Philippe de Groote, Yusuke Kubota, Robert Levine and Michael Moortgat for their discussion on the topics which led to the publication of this paper.

I would also like the referees of the Empirical Advances in Categorial Grammar (2015) workshop for the remarks and comments on the submitted abstract.

References

Aarts, Erik and Kees Trautwein. 1995. Non-associative Lambek categorial grammar in polynomial time. *Mathematical Logic Quarterly* 41:476–484.

Andreoli, Jean-Marc. 1992. Logic programming with focussing proofs in linear logic. *Journal of Logic and Computation* 2(3).

Areces, Carlos, Raffaella Bernardi, and Michael Moortgat. 2001. Galois connections in categorial type logic. In G.-J. Kruijff, L. Moss, and R. T. Oehrle, eds., *Proceedings of FGMOL 2001*, vol. 53 of *Electronic Notes in Theoretical Computer Science*, 3–20. Elsevier.

Barker, Chris and Chung-Chieh Shan. 2014. *Continuations and Natural Language*. Oxford Studies in Theoretical Linguistics. Oxford University Press.

Bernardi, Raffaella and Michael Moortgat. 2010. Continuation semantics for the lambek–grishin calculus. *Information and Computation* 208(5):397–416.

Blackburn, Patrick, Maarten de Rijke, and Yde Venema. 2001. *Modal Logic*. New York, NY, USA: Cambridge University Press. ISBN 0-521-80200-8.

Buszkowski, Wojciech. 1997. Mathematical linguistics and proof theory. In J. van Benthem and A. ter Meulen, eds., *Handbook of Logic and Language*, chap. 12, 683–736. Amsterdam: North-Holland Elsevier.

Carpenter, Bob. 1998. *Type-logical Semantics*. Cambridge, Massachusetts: MIT Press.

Danos, Vincent. 1990. *La Logique Linéaire Appliquée à l'étude de Divers Processus de Normalisation (Principalement du λ-Calcul)*. Ph.D. thesis, University of Paris VII.

de Groote, Philippe. 1999. The non-associative Lambek calculus with product in polynomial time. In N. V. Murray, ed., *Automated Reasoning With Analytic Tableaux and Related Methods*, vol. 1617 of *Lecture Notes in Artificial Intelligence*, 128–139. Springer.

de Groote, Philippe. 2001. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, 252–259. Association for Computational Linguistics.

de Groote, Philippe and Sylvain Pogodalla. 2004. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information* 13(4):421–438.

Engelfriet, Joost. 1997. Context-free graph grammars. In G. Rosenberg and A. Salomaa, eds., *Handbook of Formal Languages 3: Beyond Words*, 125–213. New York: Springer.

Girard, Jean-Yves. 1991. Quantifiers in linear logic II. In G. Corsi and G. Sambin, eds., *Nuovi problemi della logica e della filosofia della scienza*, vol. II. Bologna, Italy: CLUEB. Proceedings of the conference with the same name, Viareggio, Italy, January 1990.

Hendriks, Herman. 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics*. Ph.D. thesis, ILLC, University of Amsterdam.

Hendriks, Petra. 1995. Ellipsis and multimodal categorial type logic. In G. Morrill and R. T. Oehrle, eds., *Proceedings of Formal Grammar 1995*, 107–122. Barcelona, Spain.

Jäger, Gerhard. 2001. Anaphora and quantification in categorial grammar. In M. Moortgat, ed., *Logical Aspects of Computational Linguistics*, vol. 2014 of *Lecture Notes in Computer Science*, 70–90. Springer.

Joshi, Aravind. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. Dowty, L. Karttunen, and A. Zwicky, eds., *Natural Language Parsing*, 206–250. Cambridge University Press.

Knuth, Donald E. 2000. Dancing links. *arXiv preprint cs/0011047* .

Kubota, Yusuke and Robert Levine. 2012. Gapping as like-category coordination. In D. Béchet and A. Dikovsky, eds., *Logical Aspects of Computational Linguistics*, vol. 7351 of *Lecture Notes in Computer Science*, 135–150. Nantes: Springer.

Kubota, Yusuke and Robert Levine. 2013. Determiner gapping as higher-order discontinuous constituency. In G. Morrill and M.-J. Nederhof, eds., *Formal Grammar*, vol. 8036 of *Lecture Notes in Computer Science*, 225–241. Springer.

Kurtonina, Natasha and Michael Moortgat. 1997. Structural control. In P. Blackburn and M. de Rijke, eds., *Specifying Syntactic Structures*, 75–113. Stanford: CSLI.

Lafont, Yves. 1995. From proof nets to interaction nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, eds., *Advances in Linear Logic*, 225–247. Cambridge University Press.

Lakatos, Imre. 1978. The problem of appraising scientific theories: Three approaches. In J. Worrall and G. Currie, eds., *Mathematics, Science and Epistemology*, vol. 2 of *Philosophical Papers*, chap. 6, 107–120. Cambridge University Press.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly* 65:154–170.

Lambek, Joachim. 1961. On the calculus of syntactic types. In R. Jakobson, ed., *Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics*, vol. XII, 166–178. American Mathematical Society.

Lincoln, Patrick and Natarajan Shankar. 1994. Proof search in first-order linear logic and other cut-free sequent calculi. In *Proceedings of Logic in Computer Science (LICS'94)*, 282–291. IEEE Computer Society Press.

Martelli, Alberto and Ugo Montanari. 1982. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4(2):258–282.

Moortgat, Michael. 1996a. Generalized quantifiers and discontinuous type constructors. In H. Bunt and A. van Horck, eds., *Discontinuous Constituency*, 181–207. Berlin: Mouton de Gruyter.

Moortgat, Michael. 1996b. In situ binding: A modal analysis. In P. Dekker and M. Stokhof, eds., *Proceedings 10th Amsterdam Colloquium*, 539–549. ILLC, Amsterdam.

Moortgat, Michael. 1996c. Multimodal linguistic inference. *Journal of Logic, Language and Information* 5(3–4):349–385.

Moortgat, Michael. 1997. Categorial type logics. In J. van Benthem and A. ter Meulen, eds., *Handbook of Logic and Language*, chap. 2, 93–177. Elsevier/MIT Press.

Moortgat, Michael and Richard T. Oehrle. 1994. Adjacency, dependency and order. In *Proceedings 9th Amsterdam Colloquium*, 447–466.

Moot, Richard. 2002. *Proof Nets for Linguistic Analysis*. Ph.D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

Moot, Richard. 2007. Proof nets for display logic. Technical Report, CNRS and INRIA Futurs.

Moot, Richard. 2008. Lambek grammars and hyperedge replacement grammars. Technical Report, CNRS and Bordeaux University.

Moot, Richard. 2014a. Extended Lambek calculi and first-order linear logic. In C. Casadio, B. Coecke, M. Moortgat, and P. Scott, eds., *Categories and Types in Logic, Language, and Physics: Essays dedicated to Jim Lambek on the Occasion of this 90th Birthday*, No. 8222 in Lecture Notes in Artificial Intelligence, 297–330. Springer.

Moot, Richard. 2014b. Hybrid type-logical grammars, first-order linear logic and the descriptive inadequacy of lambda grammars. Technical Report, LaBRI (CNRS), Bordeaux University. Submitted to the IfCoLog Journal of Logic and Its Applications.

Moot, Richard. 2015. Linear one: A theorem prover for first-order linear logic. `https://github.com/RichardMoot/LinearOne`.

Moot, Richard and Mario Piazza. 2001. Linguistic applications of first order multiplicative linear logic. *Journal of Logic, Language and Information* 10(2):211–232.

Moot, Richard and Quintijn Puite. 2002. Proof nets for the multimodal Lambek calculus. *Studia Logica* 71(3):415–442.

Moot, Richard and Christian Retoré. 2012. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*. No. 6850 in Lecture Notes in Artificial Intelligence. Springer.

Moot, Richard, Xander Schrijen, Gert Jan Verhoog, and Michael Moortgat. 2015. Grail0: A theorem prover for multimodal categorial grammars. `https://github.com/RichardMoot/Grail0`.

Morrill, Glyn. 1994. *Type Logical Grammar*. Dordrecht: Kluwer Academic Publishers.

Morrill, Glyn and Oriol Valentín. 2014. Displacement logic and anaphora. *Journal of Computer and System Sciences* 80(2):390–409.

Morrill, Glyn and Oriol Valentín. 2015. Computational coverage of TLG: Displacement. In Y. Kubota and R. Levine, eds., *Empirical Advances in Categorial Grammar*. European Summer School in Logic, Language and Information.

Morrill, Glyn, Oriol Valentín, and Mario Fadda. 2011. The displacement calculus. *Journal of Logic, Language and Information* 20(1):1–48.

Muskens, Reinhard. 2001. Categorial grammar and lexical-functional grammar. In *Proceedings of the LFG01 Conference*, 259–279. University of Hong Kong.

Oehrle, Richard T. 1994. Term-labeled categorial type systems. *Linguistics & Philosophy* 17(6):633–678.

Oehrle, Richard T. 2011. Multi-modal type-logical grammar. In R. Borsley and K. Börjars, eds., *Non-transformational Syntax: Formal and Explicit Models of Grammar*, chap. 6, 225–267. Wiley-Blackwell.

Patterson, M. S. and M. N. Wegman. 1978. Linear unification. *Journal of Computing and Systems Sciences* 16:158–167.

Pentus, Mati. 1995. Lambek grammars are context free. In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, 429–433. Montreal, Canada.

Pullum, Geoffrey K. and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy* 4(4):471–504.

Shieber, Stuart. 1985. Evidence against the context-freeness of natural language. *Linguistics & Philosophy* 8:333–343.

Steedman, Mark. 2001. *The Syntactic Process*. Cambridge, Massachusetts: MIT Press.

Valentín, Oriol. 2014. The hidden structural rules of the discontinuous Lambek calculus. In C. Casadio, B. Coecke, M. Moortgat, and P. Scott, eds., *Categories and Types in Logic, Language, and Physics: Essays dedicated to Jim Lambek on the Occasion of this 90th Birthday*, No. 8222 in Lecture Notes in Artificial Intelligence, 402–420. Springer.

van Benthem, Johan. 1987. Categorial grammar and lambda calculus. In D. Skordev, ed., *Mathematical Logic and Its Applications*, 39–60. New York: Plenum Press.